Method and Apparatus for Managing a Connection Pool Using Heuristic Information

The present invention generally relates to methods and systems for managing access to computing resources. More particularly, the present invention relates to a method and system for configuring data source connection pools using heuristic information.

Background

The development of the EDVAC computer system of 1948 is often cited as the beginning of the computer era. Since that time, computer systems have evolved into extremely complicated devices. Although today's computers are more sophisticated than EDVAC, the most basic requirements levied upon computer systems have not changed. Now, as in the past, a computer system's job is to access, manipulate, and store information. This fact is true regardless of the type or vintage of computer system.

Some of the most fundamental techniques used to access, store and manipulate information are to read and write information in a data source, such as a database. Today, most databases are stored on powerful central computers called servers. Other computers, called clients, access the database by sending search requests to the server over a computer network. The server computer receives the request, performs the requested operation, and then returns the results to the client.

Although the client/server model has many advantages, one drawback is that before a client can access the database, it must first connect to that database's server. Unfortunately, managing these connections requires a significant amount of overhead. That is, the server must use a certain amount of its computing resources to create the connections, to maintain the connections, and then release connections when they are no longer required. Those skilled in the art will appreciate that the total database overhead can be particularly significant for World Wide Web ("Web") based applications because each client application will usually use one or more isolated connections to request the desired operations and because most Web clients connect and disconnect frequently. As a result, more effort is often spent connecting and disconnecting the clients than is spent processing their requests. The creation overhead can be particularly significant because it decreases the perceived speed of the Web based application.

One partial solution to this problem is to use a technique called connection pooling. In this technique, special programs called application servers create a large number of connections, in advance, so that future client requests do not need to incur the creation overhead. When the request is satisfied and the response is returned to the user, the application server returns the resource to the connection pool for reuse. In essence, this technique spreads the connection overhead across several user requests so that each user request incurs a fraction of the total creation and disconnection cost.

Although connection pooling reduces the overhead required to create and destroy connections, one problem is that the pool itself requires some overhead to maintain. Thus, if the load decreases over time, maintaining a large number of connections in connection pool uses unnecessary resources. The conventional solution is to allow idle connections to "time-out." Unfortunately, this technique does not work well for applications that do not have a static load, because, when the load again increases, the number of connections currently in the pool will not be sufficient. Connections then have to be created as required, and client applications incur this creation cost.

Without a way to manage connections for fluctuating loads, the promise of network computing may never be fully achieved.

Summary

The present invention introduces an extension to the connection pooling architecture to apply heuristic data to ensure that the connection pool contains the required number of connections for a given time period. This can improve the performance for applications that use connection pooling architectures, such as Java Database Connectivity ("JDBC") and Java 2 Connector ("J2C") connections; by attempting to predict when increased connections will be necessary, the pool can populate itself with new connections during periods of lower workload instead of consuming system resources after the workload has begun to increase. Accordingly, one aspect of the present invention is a method of configuring a server computer having a connection pool. One embodiment of this method comprises initializing a connection pool using a plurality of initial settings, generating heuristic override information, and modifying the connection pool using the heuristic override information. In some embodiments the plurality of initial settings comprises a maximum number of connections and the heuristic information may comprise a heuristic override setting and a time period.

Another aspect of the present invention is a method of operating a server. One embodiment of this method comprises initializing a connection pool with an initial maximum number of connections, applying heuristic information to modify the maximum number of connections, and in response to receiving a request to connect, detecting a current number of connections, and if the current number of connections is less than the maximum number of connections, creating a new connection.

Another aspect of the present invention is a computer program product, comprising a program configured to perform a method of operating a server and a signal bearing media bearing the program. The method of operating the server in these embodiments comprises initializing a connection pool with an initial maximum number of connections; applying heuristic information to modify the maximum number of connections; and in response to receiving a

request to connect, detecting a current number of connections, if the current number of connections is less than the maximum number of connections, creating a new connection.

Brief Description of the Drawings

Figure 1 illustrates one embodiment of an information technology system containing the present invention.

Figure 2 illustrates the operation of the one web application embodiment.

Figures 3A-3B illustrate the operation of one application server embodiment.

Figure 4 illustrates one embodiment of the heuristic configuration file.

Detailed Description

Figure 1 depicts one embodiment of an information technology system 100 comprising a web server computer 102a and a plurality of client computers 102b (only one client computer 102b shown in detail for clarity) interconnected by a communications medium 106. Each computer system 102 has one or more central processing units 110 ("CPU") connected to a main memory unit 112, a mass storage interface 114, a display interface 116, a network interface 118, and an input/output ("I/O") interface 120 by a system bus 122. The mass storage interfaces 114 connect the system busses 122 to one or more mass storage devices, such as a direct access storage device 140 and a readable and a writable optical disk drive 142. The network interfaces 118 allow the computer systems 102 to communicate with each other and to a plurality of other computers (not shown) over the communications medium 106. The main memory 112a in the web server computer 102a contains an operating system 124a, a database 126 capable of receiving and servicing JDBC database requests; an application server 128 capable of maintaining and managing a connection pool 129 and a web application 132 capable of forming and receiving JDBC database requests and capable of receiving replies from the application server 128; and a connection pool configuration file 130. The main memory 112b in the client computers 102b contains an operating system 124b, and a web browser 150 capable of communicating with the web application 132.

In operation, the connection pool configuration file 130 stores heuristic configuration information for the application server 128. The application server 128 uses the heuristic configuration information to adjust the number of connections in the pool to handle increased loads at specific times and/or specific days. In one example, the configuration file 130 for a typical e-commerce web site specifies a minimum number of connections ("MinConnections") that will be available in the pool, a maximum connection pool size ("MaxConnections") that can be contained in the pool, and a time-out period indicating how long an unused connection can remain in the pool before it is closed to reclaim resources. The configuration file 130 in this example contains the following heuristic configuration information:

```
00:00:00 - 08:00:00:  MaxConnections=0;
08:00:00 - 09:00:00:  MaxConnections=10;
09:00:00 - 11:00:00:  MaxConnections=100;
11:00:00 - 14:00:00:  MaxConnections=150;
14:00:00 - 18:00:00:  MaxConnections=200;
18:00:00 - 19:00:00:  MaxConnections=100;
19:00:00 - 20:00:00:  MaxConnections=10;
20:00:00 - 00:00:00:  MaxConnections=0;
```

The example heuristic configuration data described above indicates there is no load expected on the connection pool 129 from 00:00:00 through 08:00:00. Between 08:00:00 and 14:00:00, the load is expected to gradually increase. Between 18:00:00 and 20:00:00, the load is expected to gradually decrease. As will be discussed in more detail with reference to Figs. 2-4, when the application server 128 queries the pool size at 07:55:00, 08:55:00, 10:55:00, and 13:55:00, it will likely determine that the connection pool 129 requires additional connections and then will create those connections. When the application server 128 queries the pool size at 17:55:00, 18:55:00, and 19:55:00, it will determine that the connection pool 129 requires fewer connection and let the surplus expire. The heuristically determined number of new connections created by the application server 128 for a given time period is based on the following calculation:

(number of new connections to be created) = (number of connections specified in the configuration file 130) – (number of current connections).

Figure 2 illustrates the operation of one web application 132 embodiment in more detail, specifically a Java Database Connectivity ("JDBC") implementation of a client/server database application. At block 202, a user of the client computer 102b requests some information from the web application 132. The request requires the browser 150 to send a message to the server computer 102a at block 204 requesting that it perform some operation on the database 126 (e.g. create a new record, read an existing record, update an existing record, delete an existing record, etc). The web application 132 in this example uses the network's naming service to lookup and obtain a reference to the connection pool 129, and then transmits this message using special communication protocols called JDBC Application Programming Interfaces ("APIs") to open a connection to the database. At block 206, the web application 132 waits for the server computer 102a to perform the requested operation and return the desired information. At block 208, the web application 132 uses the JDBC APIs to close the JDBC connection to the database 126. When the web application 132 closes a connection obtained from a connection pool, the connection to the database 126 is not closed, but rather is returned to a pool of free connections. These free connections are then available to any web application 132 that requests a connection from the connection pool 129. Finally, at block 210, the web application 132 displays the requested information to its user.

Figures 3A-3B illustrate the operation of one application server 128 embodiment in more detail, specifically a JDBC implementation of a client/server database application with connection pooling. At block 302, the server computer's system administrator creates the connection pool configuration file 130, which specifies the initial number of connections contained in the pool, the minimum number of connections that will be available at all times, the maximum number of connections that can be contained in the pool, and the time-out length (i.e.. how long a connection can remain in the pool unused before being automatically closed by the connection manager). At block 304, the server administrator instructs the application server 128 to establish a pool of database connections using the ConnectionPoolDataSource API, as provided by the JDBC specification, and a set of default settings. The sever administrator then instructs the application server 128 to create a number of heuristic timer events indicating the end of each time interval defined in the configuration file 130 at block 306 and to begin normal operation at block 308. Once the application server 128 is instructed to begin normal operation, it will listen at block 309 for indication of an event, commonly called an interrupt. The application server 128 will perform certain actions after receiving an interrupt, then return to block 309 to wait for the next interrupt. Blocks 310 - 352 represent some of the interrupts to which the application server's connection pool manager will respond, together with the actions associated with that interrupt.

At blocks 310-312, if the application server 128 determines the interrupt received at block 309 is associated with a heuristic timer event, the application server 128 first compares the number of connections currently in the connection pool 129 to the MaxConnections value in the configuration file 130 associated with the current time-of-day. If the current number of connections is less than the MaxConnections associated with the current time-of-day, the application server 128 adds new connections to the pool at block 314. The application server

128 then initializes the timeout values for all available connections in the connection pool at block 316. This block ensures that none of the connections will expire until for a full time-out value after the heuristic timer event detected at block 310. The application server 128 then returns block 309 and waits for the next interrupt.

At blocks 320-322, if the interrupt received at block 309 is associated with the connection timer, the application server 128 first determines if the number of connections is at or below the minimum connection pool size. If the number of connections is above the minimum pool size, the application server 128 destroys the timed-out idle connections at block 324. That is, the connection timer interrupt is used to destroy a connection that is currently in the pool and no longer required. This closes the physical connection to the database, which frees system resources and shrinks the connection pool size by a single connection. If the number of connections is at or below the minimum number of connections, the application server 128 resets the timer for that connection at block 326.

At blocks 330-332, if the interrupt received at block 309 is an application connection request, the application server 128 first determines if the connection pool has an 'available' connection. If a connection is not available (i.e., all connections currently in the pool are being used to service other requests), the application server 128 determines at block 333 whether the number of connection currently in the pool 129 is less that the maximum number of connections specified in the configuration file 130. If the number of connections used is less than the maximum number allowed, the application server 128 creates a connection at block 335 and increments the count of the connections in the connection pool at block 336; otherwise the application server 128 waits for a connection to be returned to the connection pool at block 334. At block 338, the application server 128 uses the existing connection or the newly created connection to honor the connection request, marks the used connection as 'unavailable' to indicates that it cannot be used to honor future connection requests, and resets that connection's "time-out" value.

At blocks 340-342, if the interrupt received at block 309 is associated with a close connection request, the application server 128 returns the connection to the connection pool by marking it as 'available.'

At blocks 350-352, if the interrupt received at block 309 is associated with a shutdown request, the application server 128 closes all connections to the database and destroys the connection pool.

Figure 4 illustrates one embodiment of the heuristic configuration file 130 in more detail. The configuration file embodiment 130 in Figure 4 comprises an eXtensible Markup Language ("XML") document having a default settings section 402 and a heuristic override section 404. The default setting section 402 comprises the name of the associated database ("datasource") 406; an initial number of connections to be created in the pool ("initConnections") 408; a minimum number of connections that will be available at all times ("MinConnections") 410; a maximum number of connections that can be contained in the pool ("MaxConnections") 412; and a time-out value indicating how long an unused connection can remain in the pool before it is closed to reclaim resources ("time-out") 414. The default settings section 402 also contains a pool analysis interval 416 that specifies when the application server 128 is to analyze the pool size to determine if new connections need to be created ("poolAnalysisInterval"). For example, if this property 416 is set to '300,' the connection pool would analyze the pool size five minutes prior to each interval defined in the heuristics file. The default settings section 402 also includes a "refreshHeuristicsInterval" value 418 that indicates the how often the application server 128 is to read in the heuristics file 130 to refresh the usage data. This setting is desirable because it allows the system administrator to modify heuristic configuration file 130 to reflect new usage patterns.

The heuristic override section 404 comprises a plurality of overrides 420a-420n. Each override 420 comprises a "BeginTime" section 422a-422n, an "EndTime" section 424a-424n, and a heuristic override value 426a-426n. The BeginTime and EndTime sections 422, 424 specify a time during which the application server should use the associated heuristic override value 426. The system administrator can use these overrides to reflect the expected usage data. Some embodiments may also specify a "WeekDay" section (not shown for clarity) that will allow the system administrator to adjust based on day of week (e.g., the expected load will be high during business days, but low at night and on weekends) and a "holidays" section (not shown for clarity) that will allow the system administrator to adjust specific days (e.g., the expected load will be high the day after Thanksgiving). The application server 128 can apply these heuristics by dynamically creating additional connections prior to these peaks using the process described with reference to Figure 3. One advantage of the embodiment in Figure 4 is that the connection pools have a one-to-one association with the database 126 and the configuration file 130, which allows the server's administrator can individually configure the connection pool heuristics of the present invention for each database 126 running on the server.

Referring again to Figure 1, the computer systems 102 in this embodiment are general-purpose programmable computing devices. Accordingly, the central processing units 110 may be any device capable of executing the program instructions stored in main memory 112, and may be constructed from one or more microprocessors and/or integrated circuits. In this embodiment, when one of the computer systems 102 start up, the associated CPU 110 initially executes the program instructions that make up the operating system 124, which manages the physical and logical resources of the computer system 102. These resources include the central processing unit 110, the main memory 112, the mass storage interface 114, the display interface 116, the network interface 118, and the system bus 122. Moreover, although each computer system 102 in Figure 1 is shown to with only a single processing unit 110 and a single system bus 122, those skilled in the art will appreciate that the present invention may be practiced using a computer system 102 that has multiple processing units 110 and/or multiple system buses 122. In addition, the interfaces 114, 116, 118, and 120 may each include their own separate, fully programmed microprocessors, which may be used to off-load compute-intensive processing from the main processing units 110.

The main memory 112 and the storage devices 140, 142 may be any system capable of storing and retrieving data for the central processing units 110. These systems may utilize virtual addressing mechanisms that allow the computer systems 102 to behave as if they only have access to a large, single storage entity instead of access to multiple, smaller storage entities such as main memory 112 and DASD device 140. Therefore, while the operating systems 124, the application servers 128, and the databases 126 are shown to reside in main memory 112, those skilled in the art will recognize that these items are not necessarily all completely contained in main memory 112 at the same time, and may even reside in the virtual memory of other computer systems coupled to the computer system 102.

The display interface 116 is used to directly connect one or more display units 180 to the computer system 102. These display units 180 may be non-intelligent (i.e., dumb) terminals, such as a cathode ray tube, or may themselves be fully programmable workstations used to allow IT administrators and users to communicate with one or more of the computer systems 102. Note, however, that while the display interface 116 is provided to support communication with one or more displays 180, the computer systems 102 does not necessarily require a display 180 because all needed interaction with users and other processes may occur via network interface 118.

The communication medium 106 can be any device or system that allows the computer systems 102 to communicate with each other. The network interfaces 118, accordingly, can be any device that facilitates such communication, regardless of whether the network connection is made using present-day analog and/or digital techniques or via some networking mechanism of the future. Suitable communication mediums 106 include, but are not limited to, the Internet, intranets, cellular transmission networks, wireless networks using one of the IEEE 802.11 specifications, and the like. Those skilled in the art will appreciate that many different network protocols can be used to implement the communication medium 106. The Transmission Control Protocol/Internet Protocol ("TCP/IP") is an example of a suitable network protocol for Internet-based communication.

The embodiment described with reference to Figures 1-4 generally uses a client-server network architecture. These embodiments are desirable because the clients 102b can utilize the services of the web server computers 102a without either computer system 102 requiring knowledge of the working details about the other. However, those skilled in the art will appreciate that other network architectures are within the scope of the present invention. Examples of other suitable network architectures include peer-to-peer architectures, grid architectures, and multi-tier architectures. Accordingly, the terms web server and client computer should not be construed to limited the invention to client-server network architectures.

One suitable web server computer 102a is an eServer iSeries computer running the OS/400 multitasking operating system, both of which are produced by International Business Machines Corporation of Armonk, NY. One client computer 102b is an IBM ThinkPad running the Linux or Windows operating systems. However, those skilled in the art will appreciate that the mechanisms and apparatus of the present invention apply equally to any computer system 102 and operating system 124, regardless of whether one or both of the computer 102 are complicated multi-user computing apparatuses, a single workstations, lap-top computers, mobile telephones, personal digital assistants ("PDAs"), video game systems, or the like.

Although the present invention has been described in detail with reference to certain examples thereof, it may be also embodied in other specific forms without departing from the essential spirit or attributes thereof. For example, those skilled in the art will appreciate that the present invention is capable of being distributed as a program product in a variety of forms, and applies equally regardless of the particular type of signal bearing media used to actually carry out the distribution. Examples of suitable signal bearing media include, but are not limited to: (i) information permanently stored on non-writable storage media (e.g., read-only memory devices within a computer such as CD-ROM disks readable by a CD-ROM drive); (ii) alterable information stored on writable storage media (e.g., floppy disks within a diskette drive, a CD-R disk, a CD-RW disk, or hard-disk drive); or (iii) information conveyed to a computer by a communications medium, such as through a computer or telephone network, including wireless communications, and specifically includes information downloaded from the Internet and other networks. Such signal-bearing media, when carrying computer-readable instructions that direct the functions of the present invention, represent embodiments of the present invention. In addition, the present invention and its heuristic connection pool can be applied to other architectures that use connection pooling in a manner that is similar to JDBC connection pooling. For example, the J2C architecture provides a connection pooling mechanism that uses a common connector architecture for connection to various resources (e.g. Java Message Service (JMS) connections, connections to legacy applications such as CICS, PeopleSoft, etc.

The accompanying figures and this description depicted and described embodiments of the present invention, and features and components thereof. Those skilled in the art will appreciate that any particular program nomenclature used in this description was merely for convenience, and thus the invention should not be limited to use solely in any specific application identified and/or implied by such nomenclature. Thus, for example, the routines executed to implement the embodiments of the invention, whether implemented as part of an operating system or a specific application, component, program, module, object, or sequence of instructions could have been referred to as a "program", "application", "server", or other meaningful nomenclature. Therefore, it is desired that the embodiments described herein be considered in all respects as illustrative, not restrictive, and that reference be made to the appended claims for determining the scope of the invention.